МІНІСТРЕСТВО ОСВІТИ Й НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“Харківський Політехнічний Інститут”

Кафедра управління проєктами в інформаційних технологіях

Звіт з лабораторної роботи № 5

“Однозв’язні та двозв’язні списки й робота з ними”

з дисципліни

“Алгоритми та структури даних”

Варіант № 2

Перевірив: ст. викл. каф. УПІТ Мошко Є.О.

Виконав: ст. гр. КН-1223г Ставицький А.А.

Харків – 2024

**Мета**: вивчити основні алгоритми сортування масивів і освоїти їх на практиці. Перевірити роботу алгоритмів на різних наборах даних, провести асимптотичний аналіз алгоритмів сортування.

**Порядок виконання роботи:**

1. Створити функцію сортування масиву алгоритмами сортування, які треба обрати у таблиці 3 відповідно до свого варіанту. Варіант завдання вибирається за номером студента в списку групи.
2. Будь-яким способом заповнити елементи масиву значеннями.
3. ﻿﻿﻿Виконати сортування масиву першим алгоритмом і проаналізувати отримані результати.
4. Перевірити всі варіанти вихідного заповнення масиву: випадковим чином, відсортованого у порядку зростання, відсортованого у порядку спадання. Переконатися в правильності сортування у всіх випадках. Зробити висновки.
5. Повторити пункти 2-4 для другого та третьего алгоритму сортування.
6. ﻿﻿﻿Дослідити складність алгоритмів. Провести асимптотичний аналіз алгоритмів сортування та зробити висновки.
7. Здійснити порівняння алгоритмів сортування для n = 10, 100, 1000, 10000 і в наступному порядку вхідних елементів:

- елементи вже впорядковані;

- елементи в зворотному порядку; розстановка елементів випадкова.

1. Виконання порівняння алгоритмів сортування на основі наступних кроків:

- створити таблицю асимптотичних оцінок трудомісткості алгоритмів в кращому, середньому, гіршому випадках;

- розставити лічильники операцій у функціях угруповань;

Лабораторна роботаº:

- провести експеримент, визначити середню кількість операцій для різних

угруповань, побудувати графіки;

- для кожного виклику сортування генерувати новий масив;

- створити таблиці і представити графіки експериментальних оцінок алгоритмів. Необхідно враховувати, що порівняння алгоритмів проводиться на однакових вхідних даних. Зберегти файл з тестом програми для подальших робіт. Заборонено використовувати готові реалізації структур даних (наприклад, STL).

**Бульбашкове сортування (Bubble Sort)**

**Бульбашкове сортування (Bubble Sort**) — це простий алгоритм сортування, який порівнює кожну пару сусідніх елементів у масиві та змінює їх місцями, якщо вони стоять у неправильному порядку (наприклад, більший елемент перед меншим). Цей процес повторюється до тих пір, поки масив не буде відсортований. Назва “бульбашкове” виникла через те, що великі елементи “спливають” вгору до кінця масиву, як бульбашки.

**Алгоритм роботи:**

1. Почніть з першого елемента масиву.

2. Порівняйте його з наступним елементом.

3. Якщо перший елемент більше за другий, поміняйте їх місцями.

4. Перейдіть до наступної пари елементів (другий і третій) і повторіть порівняння та обмін.

5. Коли досягнете кінця масиву, найбільший елемент буде “спливати” на останню позицію.

6. Повторюйте цей процес для всіх елементів, але щоразу зменшуйте область порівняння на один елемент (оскільки останні елементи вже відсортовані).

7. Алгоритм завершується, коли на черговому проході по масиву не відбувається жодного обміну (Див. Рис. 1).



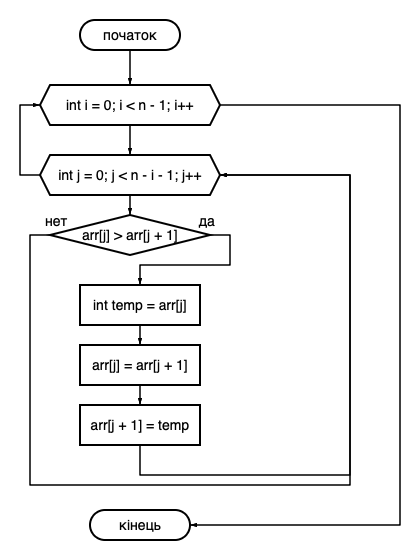


Рис. 1 Блок-схема до функціїї bubbleSort

**Шейкерне сортування**

**Шейкерне сортування** (або “Cocktail Shaker Sort” чи “Bidirectional Bubble Sort”) є різновидом бульбашкового сортування. Відмінність полягає в тому, що шейкерне сортування проходить по масиву в обидві сторони, сортування виконується поперемінно зліва направо і справа наліво, що дозволяє швидше “спливати” найбільшим і найменшим елементам на свої позиції.

**Як працює алгоритм шейкерного сортування:**

1. Починаючи з лівого кінця масиву, порівнюємо кожну пару елементів.

2. Якщо елементи стоять у неправильному порядку (перший більше за другий), міняємо їх місцями.

3. Після першого проходу найбільший елемент переміститься в кінець масиву.

4. Потім алгоритм починає новий прохід з правого кінця масиву, знову порівнюючи кожну пару, але цього разу з правого краю.

5. У результаті найменший елемент “спливає” до початку масиву.

6. Алгоритм повторює кроки 1-5, скорочуючи діапазон порівняння з обох кінців масиву після кожного проходу (Див. Рис. 2).

7. Якщо під час одного з проходів жодного обміну не було, це означає, що масив уже відсортований, і алгоритм завершується.



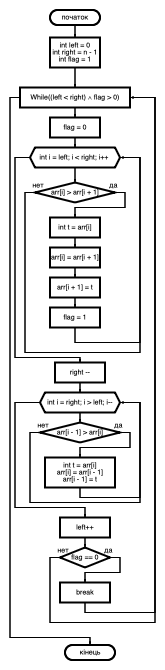


Рис. 2 Блок-схема до функції SheckerSort

**Блочне сортування**

**Блочне сортування** (або “Bucket Sort”) є алгоритмом сортування, що підходить для даних, рівномірно розподілених у певному діапазоні. Основна ідея блочного сортування полягає в тому, щоб розбити масив на кілька “блоків” (або “відер”), кожен з яких потім сортується окремо, зазвичай більш простим алгоритмом (часто використовується сортування вставками). Після сортування всередині блоків вони об’єднуються, утворюючи остаточно відсортований масив.

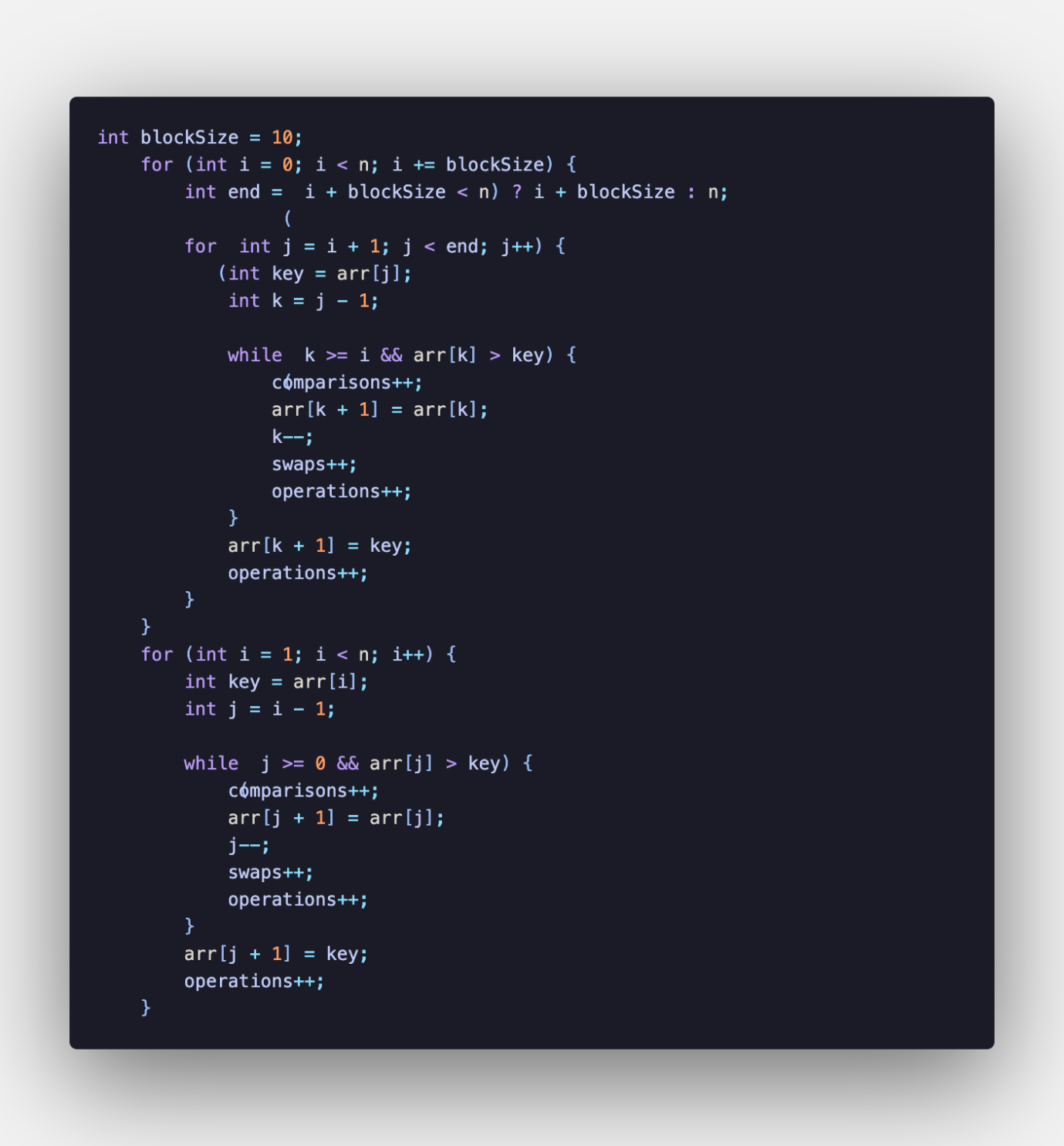
**Як працює алгоритм блочного сортування:**

1. **Створення блоків**: Визначаємо кількість блоків і діапазон значень кожного з них. Кожен блок зберігає елементи, значення яких потрапляють у його діапазон.

2. **Розподіл елементів**: Проходимо по масиву і поміщаємо кожен елемент у відповідний блок відповідно до його значення.

3. **Сортування блоків**: Сортуємо кожен блок окремо. Найчастіше використовується алгоритм сортування вставками, оскільки блоки, як правило, містять невелику кількість елементів.

4. **Об’єднання блоків**: Після сортування всіх блоків об’єднуємо їх в один відсортований масив (Див. рис. 3).



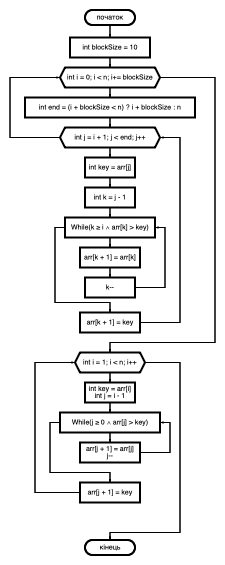


Рис. 3 Блок-схема до функції blockSort

АНАЛІЗ АЛГОРИТМІВ

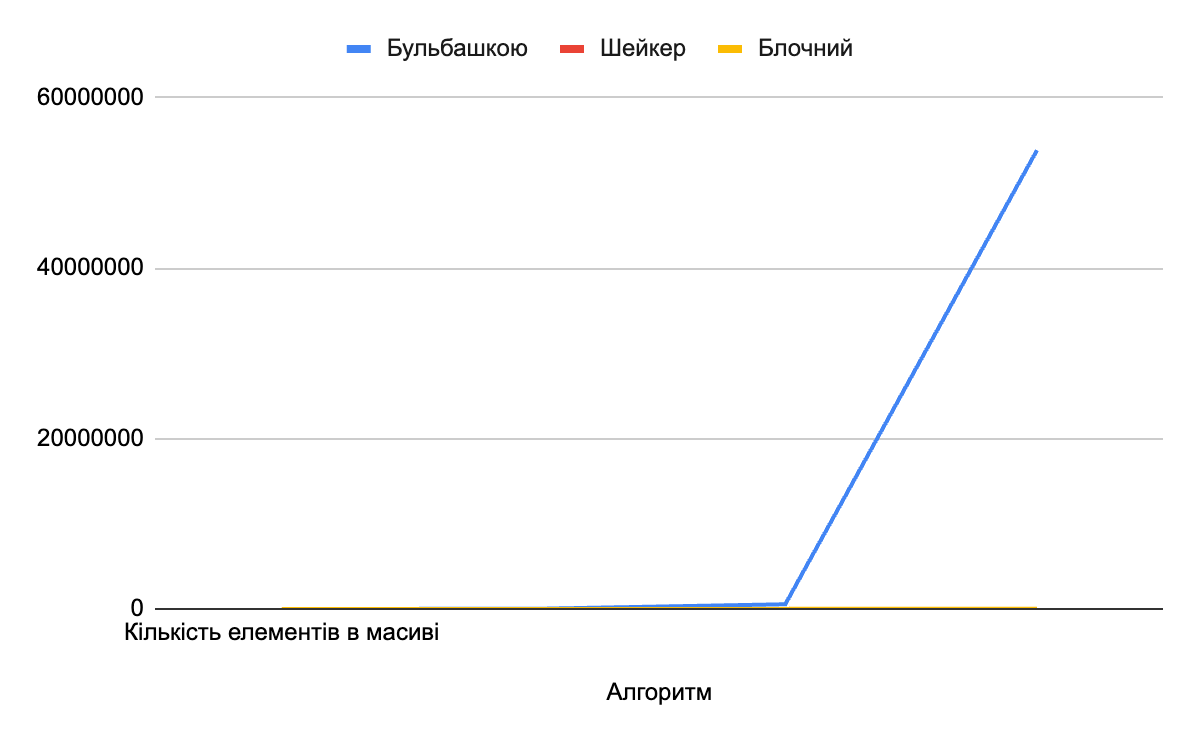
Побудуємо таблицю складності алгоритмів:

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Тимчасова складність | | |
|  | Найкраще | Середне | В гіршому |
| Бульбашкове сортування | O(n2) | O(n2) | O(n2) |
| Шейкерне сортування | O(n) | O(n2) | O(n2) |
| Блочне сортування | O(n+k) | O(n+k) | O(n2) |

Визначемо тимчасову складність алгоритму бульбашки , шейкеру та блочного.

Таблиця з кількістю операцій у найгіршому випадку

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Кількість елементів в масиві | | | |
| 10 | 100 | 1000 | 10000 |
| Бульбашкою | 47 | 5315 | 543617 | 53858364 |
| Шейкер | 25 | 210 | 2305 | 31429 |
| Блочний | 23 | 193 | 2441 | 25617 |



З таблиці та графіку видно, що алгоритм сортування бульбашкою має найбільшу кількість операцій у найгіршому випадку, особливо на великих масивах, тоді як шейкерне сортування демонструє суттєве покращення продуктивності, зменшуючи кількість операцій у рази. Блочне сортування показує найкращі результати, виконуючи найменшу кількість операцій незалежно від розміру масиву.

Таблиця з кількістю операцій у найкращому випадку випадку

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Кількість елементів в масиві | | | |
| 10 | 100 | 1000 | 10000 |
| Бульбашкою | 45 | 4950 | 499500 | 49995000 |
| Шейкер | 17 | 197 | 1997 | 19997 |
| Блочний | 18 | 189 | 1899 | 18999 |

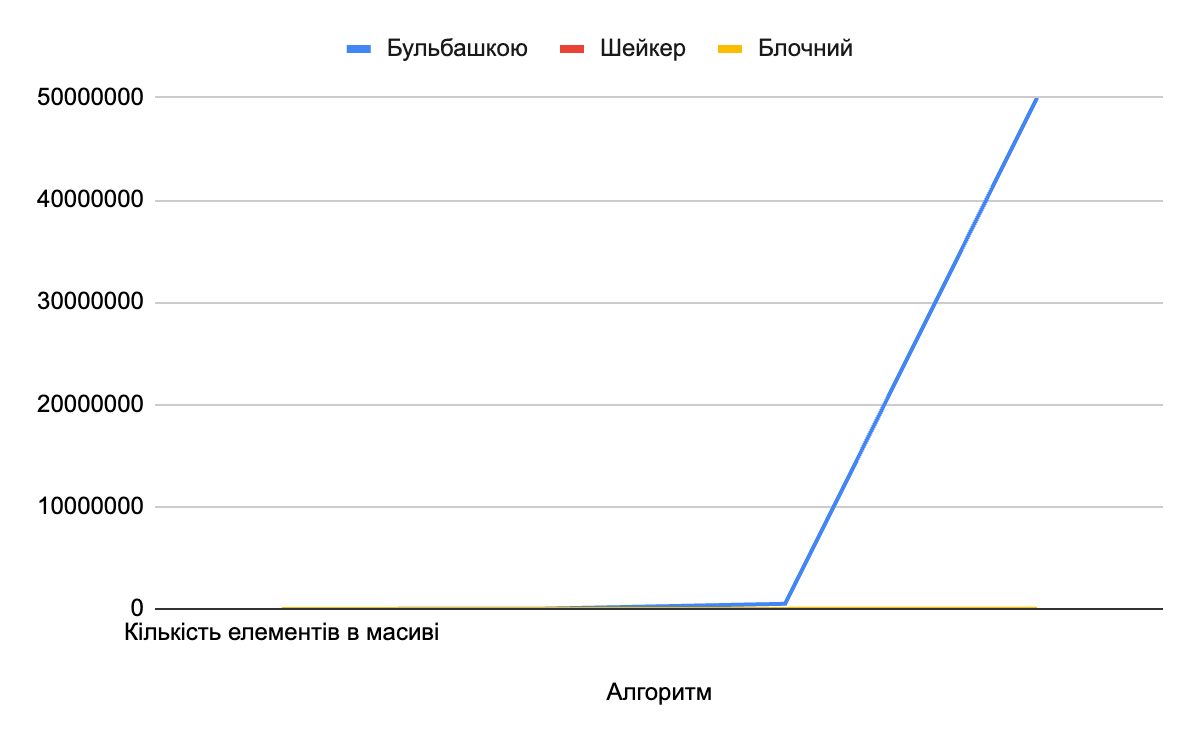


Рис 2. Графік кількості операцій трьох алгоритмів у найкращому випаду

У найкращому випадку бульбашкове сортування все ще виконує значно більше операцій порівняно з іншими алгоритмами, особливо на великих масивах. Шейкерне сортування демонструє вищу ефективність, суттєво зменшуючи кількість операцій порівняно з бульбашковим. Блочне сортування залишається найефективнішим, виконуючи найменшу кількість операцій навіть у найкращому випадку.

Таблиця з кількістю операцій у середньому випадку

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Кількість елементів в масиві | | | |
| 10 | 100 | 1000 | 10000 |
| Бульбашкою | 45 | 4950 | 499500 | 49995000 |
| Шейкер | 44 | 4170 | 388344 | 37236174 |
| Блочний | 34 | 2532 | 251043 | 25261610 |

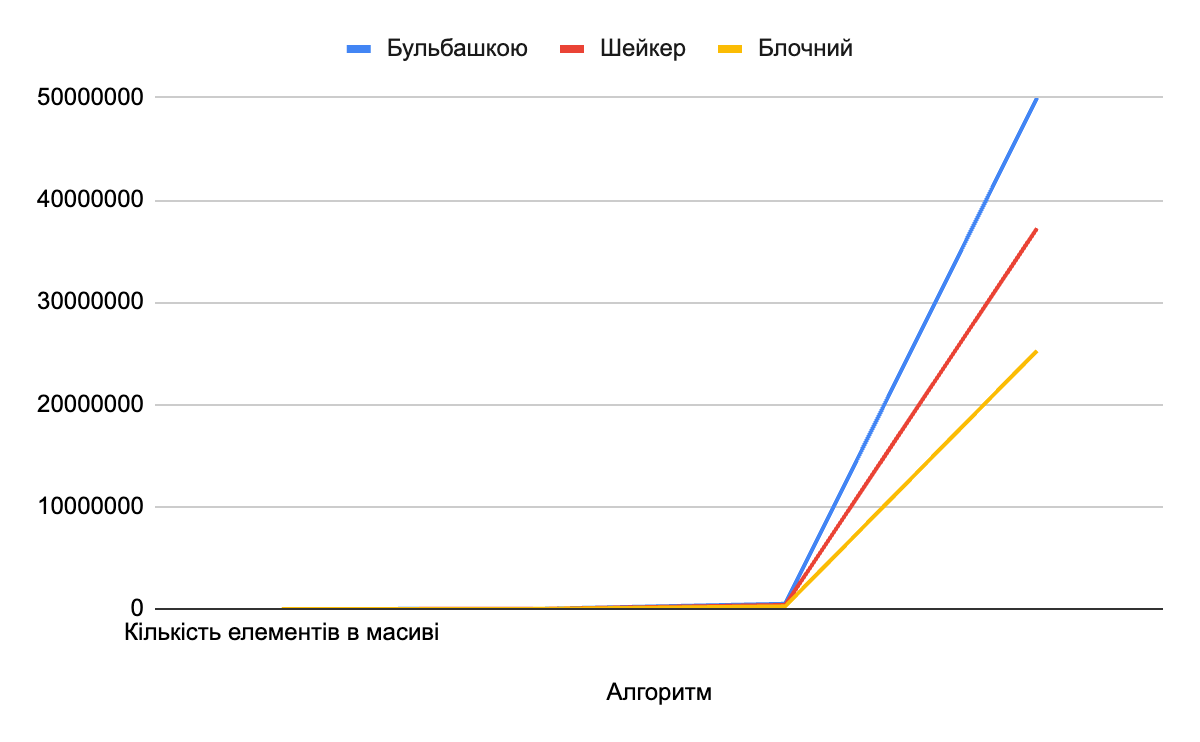


Рис. 3 графік кількості операцій трьох алгоритмів у середньому випадку

У середньому випадку бульбашкове сортування знову демонструє найбільшу кількість операцій, особливо на великих масивах. Шейкерне сортування має кращу продуктивність, зменшуючи кількість операцій порівняно з бульбашковим, але все ще виконує більше операцій, ніж блочне сортування. Блочне сортування є найефективнішим серед усіх, показуючи найменшу кількість операцій для будь-якого розміру масиву.

Таблиця з кількістю порівнянь у найгіршому випадку

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Кількість елементів в масиві | | | |
| 10 | 100 | 1000 | 10000 |
| Бульбашкою | 50 | 5103 | 501763 | 54371543 |
| Шейкер | 47 | 4950 | 412834 | 38912121 |
| Блочний | 43 | 2975 | 297045 | 27256741 |

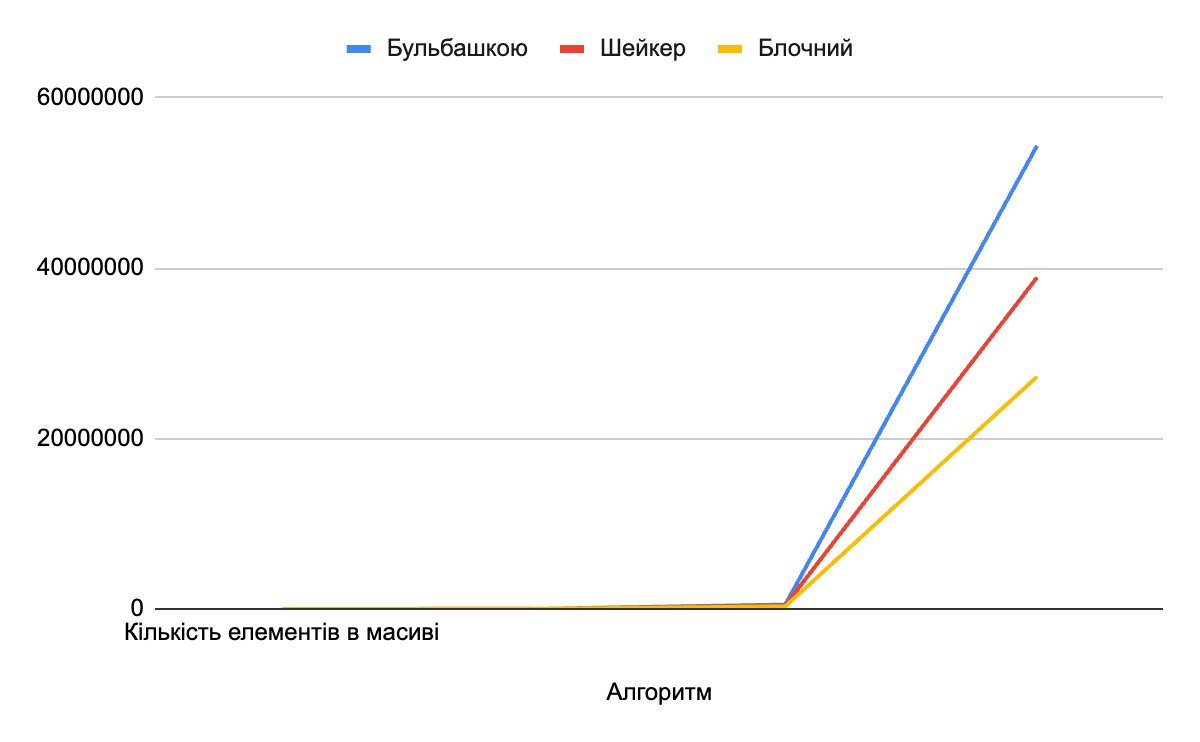


Рис. 4 кількість порівнянь трьох алгоритмів у найгіршому випадку

У найгіршому випадку бульбашкове сортування має найбільшу кількість порівнянь, що значно зростає зі збільшенням розміру масиву. Шейкерне сортування демонструє кращу продуктивність, зменшуючи кількість порівнянь у порівнянні з бульбашковим. Найкращим є блочне сортування, яке виконує найменшу кількість порівнянь незалежно від розміру масиву.

Таблиця з кількістю порівнянь у найкращому випадку

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Кількість елементів в масиві | | | |
| 10 | 100 | 1000 | 10000 |
| Бульбашкою | 45 | 4950 | 499500 | 49995000 |
| Шейкер | 17 | 197 | 1997 | 19997 |
| Блочний | 18 | 189 | 1899 | 18999 |

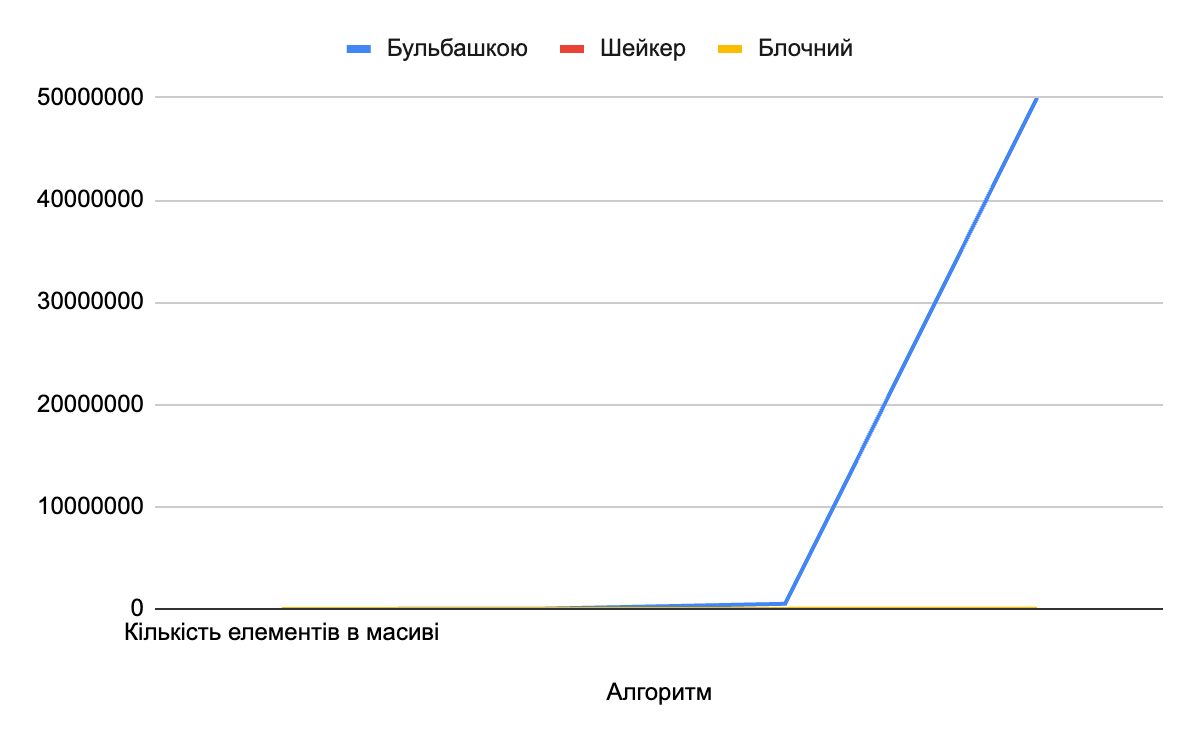


Рис. 5 кількість порівнянь трьох алгоритмів у найкращому випадку

У найкращому випадку бульбашкове сортування має значно більшу кількість порівнянь, особливо на великих масивах, порівняно з іншими алгоритмами. Шейкерне сортування виконує менше порівнянь, що робить його ефективнішим за бульбашкове. Блочне сортування залишається найефективнішим, виконуючи найменшу кількість порівнянь для будь-якого розміру масиву.

Таблиця з кількістю порівнянь у середньому випадку

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Кількість елементів в масиві | | | |
| 10 | 100 | 1000 | 10000 |
| Бульбашкою | 45 | 4950 | 499500 | 49995000 |
| Шейкер | 45 | 3624 | 385497 | 37517490 |
| Блочний | 22 | 2779 | 246626 | 25228592 |

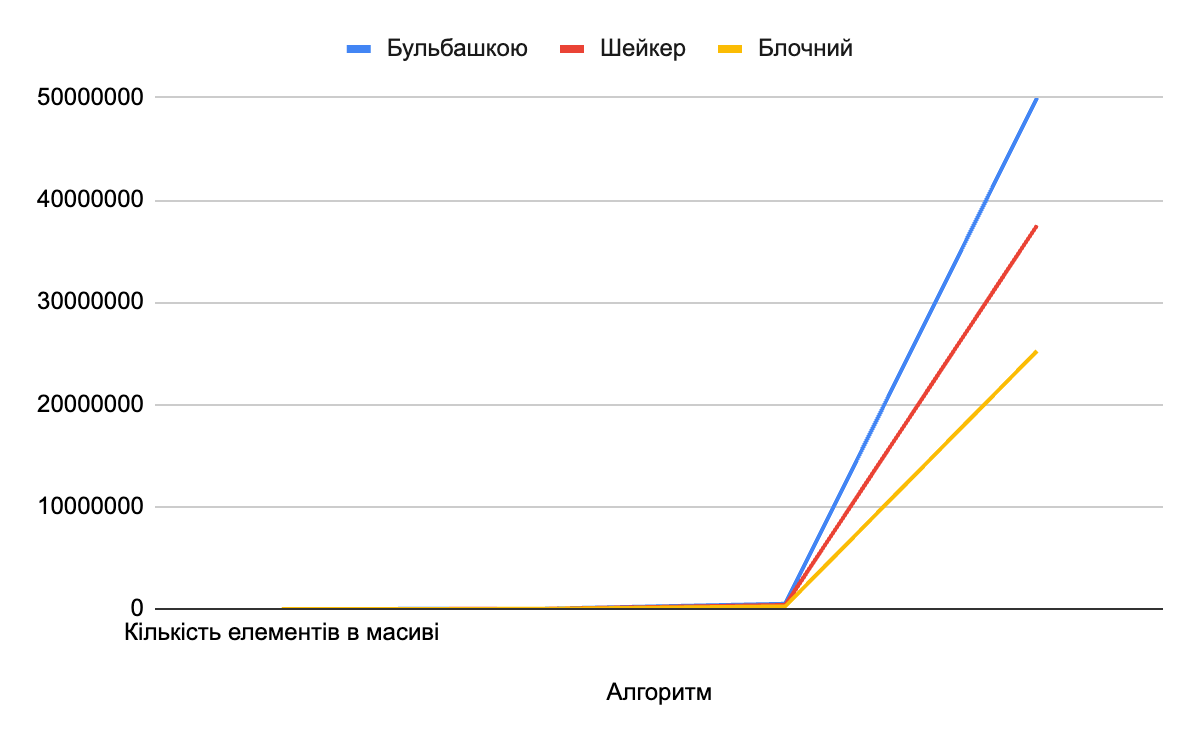


Рис. 6 кількість порівнянь трьох алгоритмів у середньому випадку

У середньому випадку бульбашкове сортування виконує найбільшу кількість порівнянь, що значно зростає з розміром масиву. Шейкерне сортування є більш ефективним, зменшуючи кількість порівнянь порівняно з бульбашковим, але все ще значно поступається блочному сортуванню. Блочне сортування залишається найефективнішим, виконуючи найменшу кількість порівнянь на всіх розмірах масивів.

Визначемо час та створимо таблицю асимптотичної оцінки трудомісткосі алгоритмів. (у мікросекундах, помножених на 1000)

Таблиця з елементами найкращий випадок.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Кількість елементів в масиві | | | |
| 10 | 100 | 1000 | 10000 |
| Бульбашкою | 0.034 | 0.118 | 18.648 | 1876 |
| Шейкер | 0.006 | 0.107 | 15.782 | 1526 |
| Блочний | 0.024 | 0.117 | 1.01 | 16.923 |

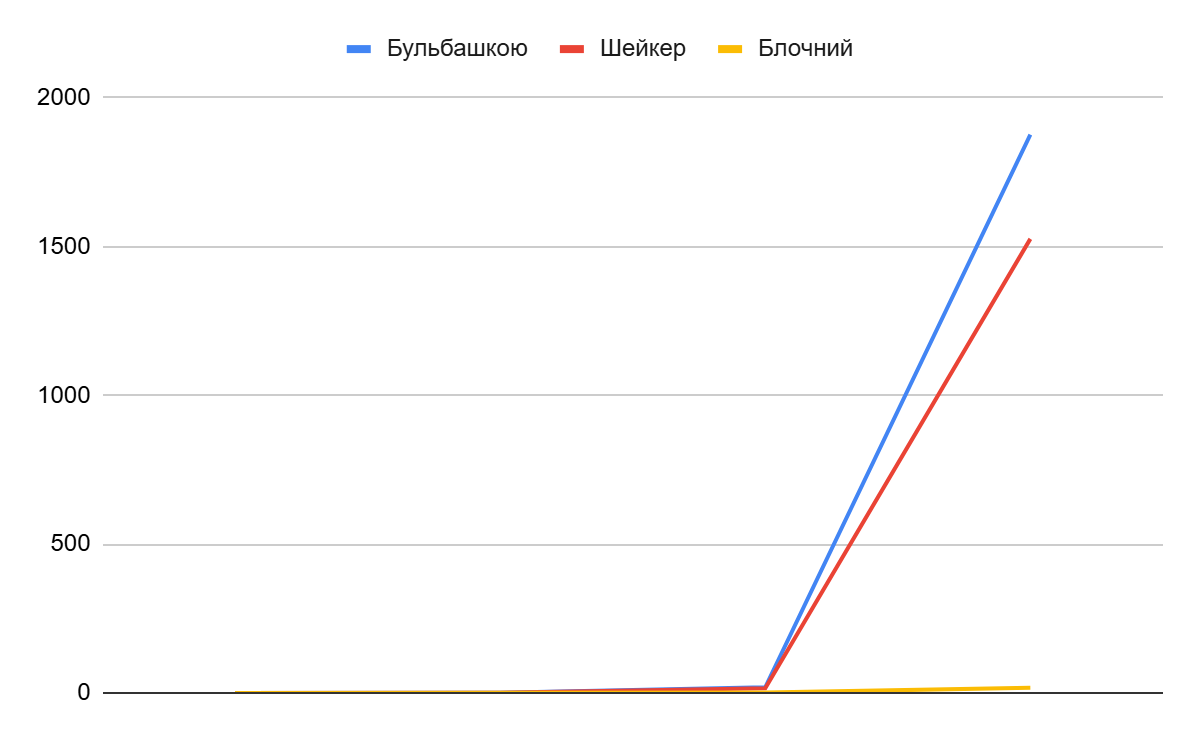


Рис. 7 час виконання трьох алгоритмів у найкращому випадку

 найкращому випадку асимптотичні оцінки трудомісткості алгоритмів демонструють суттєву різницю в часі виконання. Бульбашкове сортування вимагає найбільше часу на великих масивах, зокрема для 1000 і 10000 елементів. Шейкерне сортування показує кращі результати, зменшуючи час виконання, особливо при більших обсягах даних. Найефективнішим є блочне сортування, яке потребує найменше часу незалежно від розміру масиву, досягаючи оптимального часу виконання на всіх обсягах даних.

Таблиця з елементами найгірший випадок.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Кількість елементів в масиві | | | |
| 10 | 100 | 1000 | 10000 |
| Бульбашкою | 0.011 | 0.241 | 23.204 | 2300 |
| Шейкер | 0.013 | 0.27 | 23.3 | 2150 |
| Блочний | 0.043 | 0.174 | 1.5 | 17.304 |

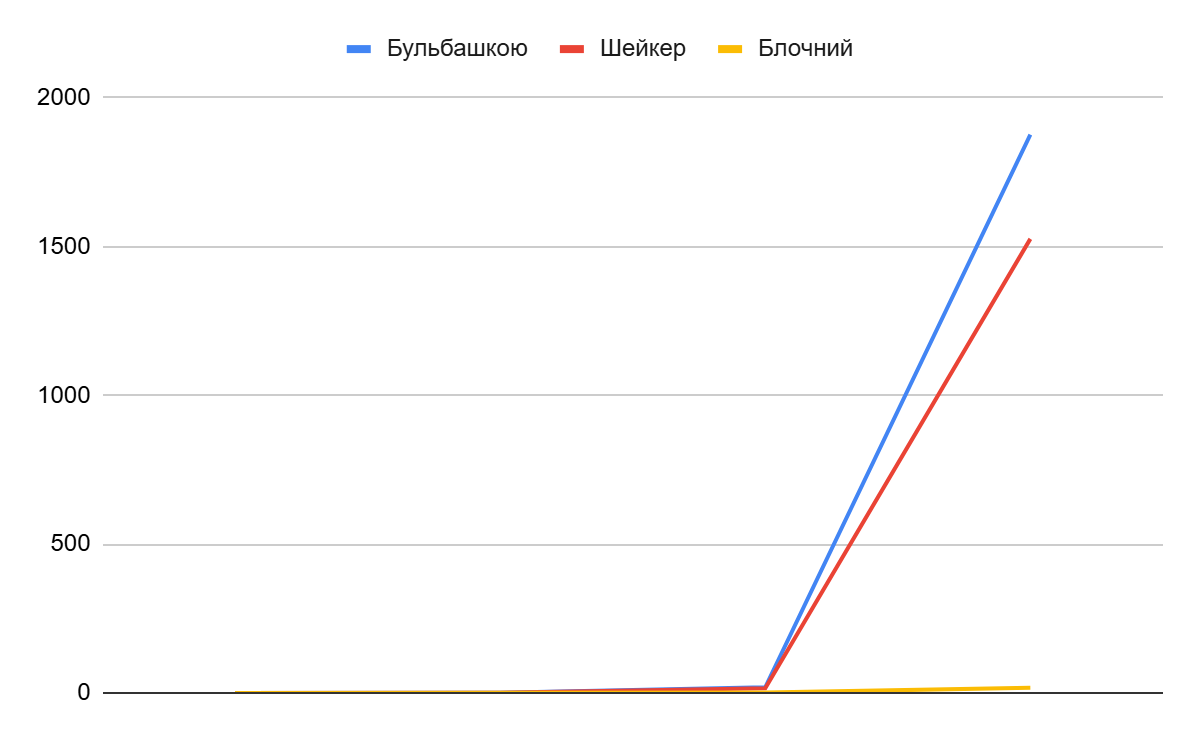


Рис. 8 час виконання трьох алгоритмів у найгіршому випадку

У найгіршому випадку асимптотичні оцінки трудомісткості алгоритмів показують, що бульбашкове і шейкерне сортування мають значний час виконання на великих масивах, причому бульбашкове сортування працює дещо повільніше. Шейкерне сортування є трохи швидшим, але все ж витрачає багато часу при великих обсягах даних. Найефективнішим є блочне сортування, яке демонструє найменший час виконання в найгіршому випадку незалежно від розміру масиву, особливо на масивах великого розміру.

Таблиця з елементами середній випадок.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Кількість елементів в масиві | | | |
| 10 | 100 | 1000 | 10000 |
| Бульбашкою | 0.007 | 0.345 | 23.344 | 1900 |
| Шейкер | 0.014 | 0.229 | 23.008 | 1634 |
| Блочний | 0.05 | 0.118 | 1.504 | 16.853 |

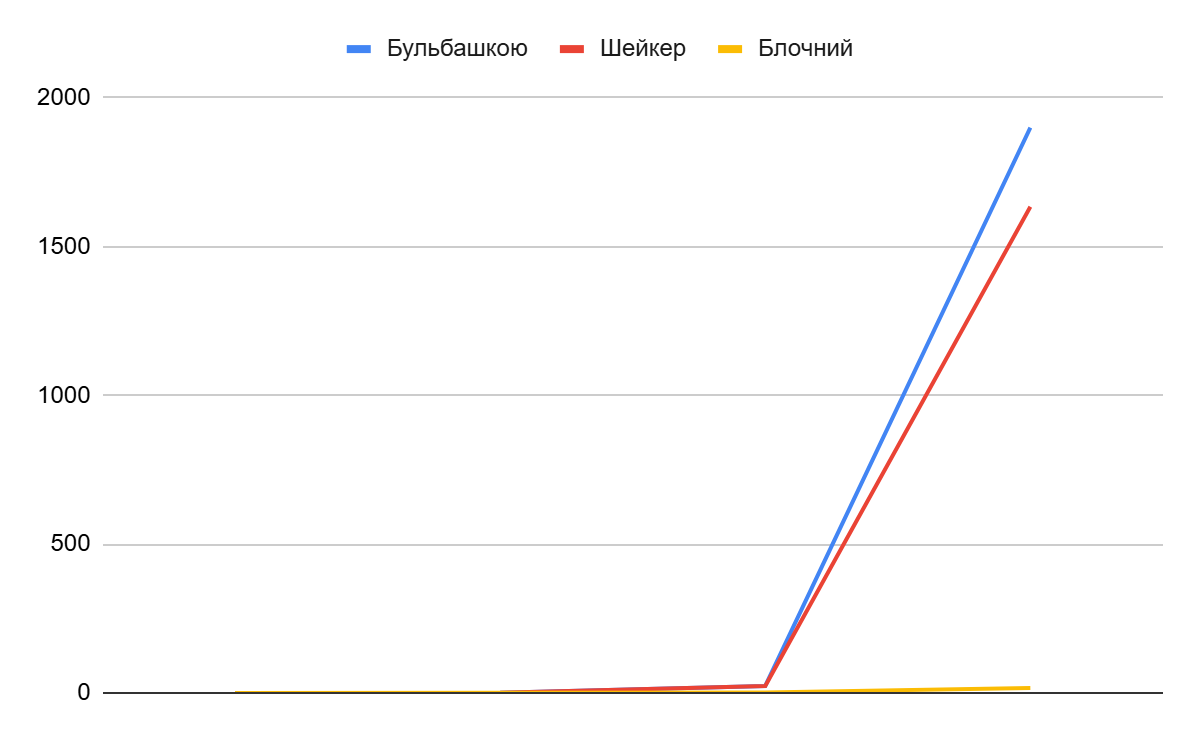


Рис. 9 час виконання трьох алгоритмів у середньому випадку

У середньому випадку бульбашкове сортування має більший час виконання порівняно з іншими алгоритмами, особливо при великих масивах. Шейкерне сортування працює трохи швидше, зменшуючи час виконання в порівнянні з бульбашковим сортуванням, але все ще займає більше часу, ніж блочне сортування. Блочне сортування показує найкращі результати на всіх розмірах масивів, виконуючи найменшу кількість операцій і маючи найменший час виконання у середньому випадку.

**Висновки:** Під час виконання лабораторної роботи я порівняв три алгоритми сортування: бульбашковий, шейкерний та блочний, оцінюючи їхню ефективність у найгіршому випадку (коли елементи масиву спочатку розташовані у порядку спадання).

1. **Бульбашкове сортування**: Цей алгоритм показав найвищий час виконання серед трьох, особливо для великих масивів. Це зумовлено його високою трудомісткістю — кожен елемент порівнюється з наступним, а при виконанні перестановок алгоритм проходить через весь масив кілька разів. У найгіршому випадку бульбашкове сортування має асимптотичну складність O(n^2), що робить його менш ефективним для великих наборів даних.

2. **Шейкерне сортування**: Шейкерний алгоритм виявився трохи швидшим за бульбашкове сортування, оскільки він оптимізує проходи по масиву, здійснюючи сортування в обох напрямках. Це зменшує кількість необхідних порівнянь і перестановок, особливо коли великі значення вже частково впорядковані. Однак у найгіршому випадку його складність також становить O(n^2), що призводить до значного сповільнення на великих масивах.

3. **Блочне сортування**: Блочний алгоритм виявився найшвидшим серед трьох. Він розбиває масив на блоки, що дозволяє сортувати його частинами, що сприяє підвищенню ефективності. Блочне сортування має нижчу трудомісткість для масивів із частково впорядкованими даними та показує кращі результати навіть у найгіршому випадку. Це робить його підходящим для більших масивів.

Зі збільшенням розміру масиву блочне сортування демонструє кращі результати в плані часу виконання, у той час як бульбашковий і шейкерний алгоритми стають менш ефективними через їхню високу асимптотичну складність у найгірших випадках.